Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 23 April 2026

R. White
Akamai
S. Hegde
T. Przygienda
Juniper Networks
L. Jalil
Verizon
D. Voyer
Cisco

20 October 2025

IS-IS Distributed Flooding Reduction
draft-ietf-lsr-distoptflood-11

Abstract

In dense topologies (such as data center fabrics based on the Clos and butterfly though not limited to those; in fact any large topology or one with relatively high degree of connectivity qualifies here) IGP flooding mechanisms designed originally for rather sparse topologies can "overflood", or in other words generate too many identical copies of same information arriving at a given node from other devices. This normally results in longer convergence times and higher resource utilization to process and discard the superfluous copies. Flooding algorithm extensions that restrict the amount of flooding performed can be constructed and can reduce resource utilization significantly, while improving convergence performance.

One such flooding modification (based on previous art) optimized for operational considerations, described further in Section 2, is described in this document.

This Internet-Draft will expire on 23 April 2026.

Table of Contents

1. MANET-Based, Load-Balancing Flooding Extension

The following section describes a distributed algorithm similar to and based on those implemented in OSPF to support mobile ad-hoc networks, as described in [RFC5449],[RFC5614]. These solutions have been widely implemented and deployed.

1.1. Empirical Evidence of Correctness and Efficiency Improvement

Laboratory tests based on a well known open source codebase show that modifications similar to the algorithm presented here reduce flooding in a large scale emulated butterfly network topology significantly. Under unmodified flooding procedures intermediate systems receive, on average, 40 copies of any changed LSP fragment in a 2'500 nodes butterfly network. With the changes described in this document said systems received, on average, two copies of any changed LSP fragment. In many cases, only a single copy of each changed LSP was received and processed per node. In terms of performance, overall convergence times were cut in roughly half. Other topologies under experimentation in CLOS networks using another implementation show similar performance and simulations of the extension indicate significant reductions in flooding volumes.

An early version of mechanisms described here has been implemented in the FR Routing open source routing stack as part of 'fabricd' daemon and the described modification has been implement by commercial vendors.

1.2. Flooding Modifications

This section describes detailed modifications to the IS-IS flooding process to reduce the full topology to a dominating connected set of links used for flooding. It does at the same time balance the remaining flooding across all links in the topology to prevent hot-spots.

1.2.1. Example Network

Following spine and leaf fabric will be used in further description of the introduced modifications.

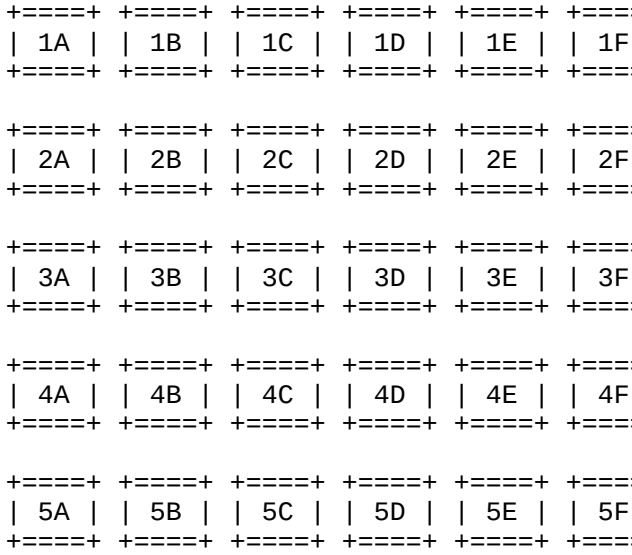

Figure 1

The above picture does not contain the connections between devices for readability purposes. The reader should assume that each device in a given layer is connected to every device in the layer above it in a butterfly network fashion. For instance:

* 5A is connected to 4A, 4B, 4C, 4D, 4E, and 4F

* 5B is connected to 4A, 4B, 4C, 4D, 4E, and 4F

* 4A is connected to 3A, 3B, 3C, 3D, 3E, 3F, 5A, 5B, 5C, 5D, 5E, and 5F

* 4B is connected to 3A, 3B, 3C, 3D, 3E, 3F, 5A, 5B, 5C, 5D, 5E, and 5F

* etc.

The tiers or stages of the fabric are marked for easier reference. Alternate representation of this topology is a "folded Clos" with T2 being the "top of the fabric" and T0 representing the leaves.

1.2.2. Optimizing Flooding

The simplest way to conceive of the solution presented here is in two stages:

* Stage 1: Forward Optimization

  - Find the group of intermediate systems that will all flood to the same set of neighbors as the local IS

  - Decide (deterministically) which subset of the intermediate systems within this group should re-flood any received LSPs

* Stage 2: Reverse Optimization

  - Find neighbors on the shortest path towards the origin of the change

  - Do not flood towards these neighbors

The first stage is best explained through an illustration. In the network above, if 5A transmits a modified Link State Protocol Data Unit (LSP) to 4A-4F, each of 4A-4F nodes will, in turn, flood this modified LSP to 3A (for instance). With this, 3A will receive 6 copies of the modified LSP, while only one copy is necessary for the intermediate systems shown to converge on the same view of the topology. If 4A-4F could determine that all of them will all flood identical copies of the modified LSP to 3A, it would be possible for all of them except one to decide not to flood the changed LSP to 3A.

The technique used in this draft to determine such flooding group is for each intermediate system to calculate a special SPT (shortest-path spanning tree) from the point of view of the transmitting neighbor. As next step, by setting the metric of all links to 1 and truncating the SPT to two hops, the local IS can find the group of neighbors it will flood any changed LSP towards and the set of intermediate systems (not necessarily neighbors) which will also flood to this same set of neighbors. If every intermediate system in the flooding set performs this same calculation, they will all obtain the same flooding group.

Once such a flooding group is determined, the members of the flooding group will each (independently) choose which of the members should re-flood the received information. A common hash function is used across a set of shared variables so each member of the group comes to the same conclusion as to the designated flooding nodes. The group member which is in such a way ‘selected‘ to flood the changed LSP does so normally; the remaining group members suppress the flooding of the LSP initially.

Each IS calculates the special, truncated SPT separately, and determines which IS should flood any changed LSPs independently based on a common hash function. Because these calculations are performed

using both a view of the network shared by all nodes (based on the common link state database) and additionally a common hash function, each member of the flooding group will make the same decision under converged conditions. In the transitory state of nodes having potentially different view of topologies the flooding may either overflood or in worse case not flood enough for which we introduce a 'quick-patching' mechanism later but ultimately will converge due to periodic CSNP origination per normal protocol operation.

The second stage is simpler, consisting of a single rule: do not flood modified LSPs along any of the shortest paths towards the origin of the modified LSP. This rule relies on the observation that any IS between the origin of the modified LSP and the local IS should receive the modified LSP from some other IS closer to the source of the modified LSP. It is worth to observe that if all the nodes that should be designated to flood within a peer group are pruned by the second stage the receiving node is at the 'tail-end' of the flooding chain and no further flooding will be necessary. Also, per normal protocol procedures flooding to the node from which the LSP has been received will not be performed.

1.2.3. Optimization Process Details

This section provides normative description of the specification. Any node implementing this solution MUST exhibit external behavior that conforms to the algorithms provided.

Each intermediate system will determine whether it should re-flood LSPs as described below. When a modified LSP arrives from a Transmitting Neighbor (TN), the result of the following algorithm obtains the necessary decision:

Step 1: Build the Two-Hop List (THL) and Remote Neighbor's List (RNL) of nodes:

A) Set all link metrics to 1

B) Calculate an SPT truncated to 2 hops from the perspective of TN. The calculation MUST consider any adjacency to a neighbor via a P-Node as a single hop and hence not include P-Nodes on THL or RNL. Usual bidirectional checks MUST be performed during the construction of SPT.

C) For each IS that is two hops away (has a metric of 2 in the truncated SPT) from TN:

   i. If the IS is the LSP originator, skip

ii. If the IS is a neighbor of the LSP originator, skip

iii. If the IS is on the shortest path from the TN towards the originator of the modified LSP, skip

iv. If the IS is *not* on the shortest path from the TN towards the originator of the modified LSP, add it to THL

D) Add each IS that is one hop away from TN to the RNL

Step 2: Sort nodes in RNL by system IDs, from the least value to the greatest.

Step 3: Use the double hashing algorithm specified in Section 1.2.4 to calculate a number H.

The shifting of the fragment number in the hashes will put 8 sequent fragments onto the same flooding subgraph to minimize re-ordering. Double hashing tries to achieve a good uniform distribution of hash values in presence of numbering schemes offering little entropy in system IDs.

RNum is the number of nodes in the RNL. Consequently, set N to the H MOD of RNum (N=H MOD RNum). With that N will be less than the number of members of RNL. (footnote 1: this allows for some balancing of LSPs coming from same system ID).

Step 4: Starting with the Nth member of RNL: where N is the index into the members in RNL, with index starting from zero (index value zero is assigned to the IS with lowest system-id):

A) If THL is empty or the walk wrapped around, move to Step 5

B) If this member of RNL is the local calculating IS, it MUST re-flood the modified LSP to at least the remaining members in the THL it is adjacent to; move to Step 5

C) If this member of the RNL signals capability to run or is running another flood reduction extension move to the next member of RNL

D) Remove all members of THL connected to (adjacent to) this member of RNL

E) Move to the next member of RNL, wrapping to the beginning of RNL if necessary

This description is leaning towards clarity rather than optimal performance when implemented.

1.2.4. Reference Hash Implementation

This section provides Rust code defining the hashing every node deploying this algorithm MUST use. This avoids the verbal explanation of the hash function with the often arising ambiguities.

```
     fn flood_reduction_hash(systemid: [u8; 6], fragment_nr: u8) -> u32 {
         fn log2(v: u8) -> u8 {
             let r = v.trailing_zeros() as _;
             assert_eq!(v, 1u8 << r);
             r
         }

         // reduce fragment number to ignore lowest bits
         let fragment_downshift = [((fragment_nr & 0xff) >> log2(8)) as u8];

         // two iterators basically reversing order of hashing values in
         let hashiters: Vec<Box<dyn Iterator<Item=(usize, &u8)>>> = vec![
             Box::new((0..).step_by(2).zip(fragment_downshift.iter().chain(systemid.it
er()))),
             Box::new((0..).step_by(5).zip(systemid.iter().rev().chain(fragment_downsh
ift.iter()))),
         ];

         // compute same type of hash over all iterators, fold the hash value on itsel
f
         // and then XOR all the hashes together again
         hashiters
             .into_iter()
             .map(move |mut iter| iter
                 .fold(0u32,
                       |previous, (offset, value)|
                           (previous << 4) ^ (offset as u32 + (*value as u32))))
             .map(move |hash| hash ^ (hash >> 14))
             .fold(0, |previous, hash| previous ^ hash)
     }
```

Figure 2

The resulting hash has been designed to fit into an unsigned 32-bit integer. When the hash is calculated the following reference results are expected. The "modulo Rnum" result is also provided for further reference.

```
ID: 01 02 03 04 05 06 fragment: 00 computed 0x0699B13A modulo %(2,3,4,5,6,7) [0, 1, 2, 4,
 4, 0]
ID: 01 02 03 04 05 06 fragment: 07 computed 0x0699B13A modulo %(2,3,4,5,6,7) [0, 1, 2, 4,
 4, 0]
ID: 01 02 03 04 05 06 fragment: 08 computed 0x0799B53B modulo %(2,3,4,5,6,7) [1, 1, 3, 0,
 1, 4]
ID: 00 01 02 03 04 05 fragment: 07 computed 0x05B99999 modulo %(2,3,4,5,6,7) [1, 1, 1, 1,
 1, 6]
ID: 01 02 03 04 05 07 fragment: 08 computed 0x0699B13A modulo %(2,3,4,5,6,7) [0, 1, 2, 4,
 4, 0]
ID: FF 05 04 03 02 01 fragment: 00 computed 0x1165D6C5 modulo %(2,3,4,5,6,7) [1, 1, 1, 4,
 1, 5]
ID: FF 05 04 03 02 01 fragment: FE computed 0x0E65AAE6 modulo %(2,3,4,5,6,7) [0, 2, 2, 0,
 2, 6]
ID: FF 05 04 03 02 01 fragment: FD computed 0x0E65AAE6 modulo %(2,3,4,5,6,7) [0, 2, 2, 0,
 2, 6]
ID: 1F 48 00 00 00 07 fragment: 00 computed 0x0506D336 modulo %(2,3,4,5,6,7) [0, 0, 2, 1,
 0, 5]
ID: 1F 48 00 00 00 07 fragment: 0F computed 0x0406D737 modulo %(2,3,4,5,6,7) [1, 1, 3, 0,
 1, 0]
ID: 01 00 00 00 00 00 fragment: 00 computed 0x006E8CA8 modulo %(2,3,4,5,6,7) [0, 1, 0, 3,
 4, 3]
ID: 01 00 00 00 00 00 fragment: 08 computed 0x016E88A9 modulo %(2,3,4,5,6,7) [1, 2, 1, 1,
 5, 3]
```

Figure 3

1.2.5. Flooding Failures

It is possible that during initial convergence or in some failure modes the flooding will be incomplete due to the optimizations outlined. Specifically, if a re-flooder fails, or is somehow disconnected from all the links across which it should be re-flooding, an LSP could be only partially distributed through the topology. To speed up convergence under such partition failures (observe that periodic CSNPs will under any circumstances converge the topology though at a slower pace), an intermediate system which does not re-flood a specific LSP (or fragment) SHOULD:

A) Set a short, configurable timer which should be significantly shorter than CSNP interval used.

B) When the timer expires, send Partial Sequence Number Packet (PSNP) of all LSPs that have *not* been re-flooded during the timer runtime to all neighbors unless an up-to-date PSNP or CSNP has been already received from the neighbor.

C) Per normal protocol procedures process any Partial Sequence Number Packets (PSNPs) received that indicate that neighbors still have older versions of the LSP will lead to the usual synchronization of the databases that are out of sync due to optimized flooding.

During processing of received CSNPs or PSNPs indicating requests for a newer version the resulting flooding MUST NOT try to use optimized flooding distribution.

If any such re-synchronizations above a configurable threshold is required (i.e. PSNPs are sent to the neighbors and are answered with requests or CSNPs are indicating significant misses of distributed LSPs), an implementation SHOULD notify the network operator via the according mechanism about the condition.

1.2.6. Deployment Considerations

A node deploying this algorithm on point-to-point links MUST send CSNPs on such links. This does not represent a dramatic change given most deployed implementations today already exhibit this behavior to prevent possible slow synchronization of IS-IS database across such links and to provide additional periodic consistency guarantees.

1.2.7. Implementation Considerations

The calculations described here seem complex, which might lead the reader to conclude that the cost of calculation is so much higher than the cost of flooding that this optimization is counter-productive. First, The description provided here is designed for clarity rather than optimal calculation. Second, many of the involved calculations can be easily performed in advance and stored, rather than being performed for each LSP occurrence and each neighbor. Optimized versions of the process described here have been implemented, and do result in strong convergence speed gains.

Further, a node that decides that the calculations involved may generate an excessive burden during massive topology fluctuations events, and may negatively affect overall protocol behavior, can decide to not apply reduction rules temporarily for any set of LSPs and fully flood those until the conditions abate.

An implementation in a node MAY choose independently of others to provide a configurable parameter to allow for more than one node in RNL to re-flood, e.g. it may re-flood even if it’s only the member that would be chosen from the RNL if a double coverage of THL is required. The modifications to the algorithm are simple enough to not require further text.

An implementation should pay particular attention that the case of a stale LSP with a higher version that persists in the network still works correctly in case the originator reboots and starts with lower version. Though the flooding of an LSP back to originator is suppressed by this extension the normal PSNP and CSNP procedures should trigger re-origination by the source of a higher version correctly.

2. Operational Considerations

The extension introduced to flooding in this document exhibits many desirable properties important for large production IS-IS networks.

This extension is designed to be deployable without initial configuration and balances the reduced flooding not only on a single or few trees but uses the information about the origin of the fragment to spread the load across whole topology.

3. Security Considerations

This document outlines flooding algorithm modification to the IS-IS protocol for operation most useful at large scale or in high density network topologies. The extension does not present any new attack vectors even if nodes start to advertise a byzantine attack of signalling that they run the extension while still following standard behavior. As always, ISIS implementations SHOULD implement IS-IS cryptographic authentication, as described in [RFC5304], and should enable other security measures in accordance to best common practices for the IS-IS protocol.

4. IANA Section

IANA is requested to allocate a TBD value in 1-127 range with description of "Modified MANET Reduction" further referring to this document as specification in the "IGP Algorithm Type For Computing Flooding Topology" registry.

5. Contributors

The following people have contributed to this draft or provided valuable comments and are mentioned without any particular order: Abhishek Kumar, Nikos Triantafillis, Ivan Pepelnjak, Christian Franke, Hannes Gredler, Les Ginsberg, Naiming Shen, Uma Chunduri, Nick Russo, Tony Li, Acee Lindem and Rodny Molina.

6. Normative References

7. Informative References

[RFC5304] Li, T. and R. Atkinson, "IS-IS Cryptographic Authentication", RFC 5304, DOI 10.17487/RFC5304, October 2008, <https://www.rfc-editor.org/info/rfc5304>.

[RFC5449] Baccelli, E., Jacquet, P., Nguyen, D., and T. Clausen, "OSPF Multipoint Relay (MPR) Extension for Ad Hoc Networks", RFC 5449, DOI 10.17487/RFC5449, February 2009, <https://www.rfc-editor.org/info/rfc5449>.

[RFC5614] Ogier, R. and P. Spagnolo, "Mobile Ad Hoc Network (MANET) Extension of OSPF Using Connected Dominating Set (CDS) Flooding", RFC 5614, DOI 10.17487/RFC5614, August 2009, <https://www.rfc-editor.org/info/rfc5614>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

Appendix A. Flooding Example

Assume, in the network specified in example given in Section 1.2.1, that 5A floods some modified LSP towards 4A-4F and we only use a single node to re-flood. To determine whether 4A should flood this LSP to 3A-3F:

* 5A is TN; 4A calculates a truncated SPT from 5A’s perspective with all link metrics set to 1

* 4A builds THL, which contains 3A, 3B, 3C, 3D, 3E, 3F, 5B, 5C, 5D, 5E and 5F

* 4A builds RNL, which contains 4A,4B,4C,4D,4E and 4F, sorting it by the system ID

* 4A computes hash on the received LSP-ID to get N; assume N is 1 in this case

* Since 4A is the 1st member of RNL and there are members in THL, 4A must re-flood; the loop exits

Authors’ Addresses

Russ White
Akamai
Email: russ@riw.us

Shraddha Hegde
Juniper Networks
Email: shraddha@juniper.net

Tony Przygienda
Juniper Networks
Email: prz@juniper.net

Luay Jalil
Verizon
Email: luay.jalil@verizon.com

Daniel Voyer
Cisco
Email: ?@cisco.com